JATIKOM: Jurnal Aplikasi dan Teori Ilmu Komputer
ISSN: 2615-577X (Online)

Evaluasi Komparatif Implementasi Finite State Automata dan Logika Kontrol Konvensional dalam Arsitektur Game Berbasis Three.js

Firas Atqiya[1], Muhammad Rizqi Sholahuddin[2]

[1]*Program Studi Teknik Informatika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Padjadjaran*
Gedung Ilmu Komputer (Gedung A PPBS), Jl. Ir. Soekarno KM 21, Jatinangor, Kabupaten Sumedang, Jawa Barat
[1]firas.atqiya@unpad.ac.id

[2]*Jurusan Teknik Komputer dan Informatika, Politeknik Negeri Bandung*
Jl. Gegerkalong Hilir, Ciwaruga, Kec. Parongpong, Kabupaten Bandung Barat, Jawa Barat, Indonesia
[2]muhammad.rizqi@polban.ac.id

Abstract—***Agent artificial intelligence logic and non-player characters in web-based games must share computational resources with graphics rendering processes, skeletal animation, and WebGL draw command execution. Both the Finite State Automata (FSA) paradigm using the State Pattern and conventional imperative control logic are commonly employed to organize this control logic; however, their architectural and computational trade-offs are rarely evaluated in a controlled manner. This study evaluates the trade-offs in computational performance, code structure quality, scalability, and behavioural determinism between both paradigms on the Three.js game architecture. Controlled experiments were conducted using a 3D arena and identical shared perception stimuli (Euclidean distance and Raycaster LoS); agents were formalized as $M = (Q, \Sigma, \delta, q_0)$ with five discrete behaviours implemented via the State Pattern featuring enter()/update()/exit() hooks, compared against a monolithic imperative controller. Performance was measured using an isolated timer strictly wrapping the controller decision update function, alongside aggregate script duration, frame time, and rendering phase duration, accompanied by Babel AST static analysis and paired inferential testing across a range of $N = 10–500$ agents (200 sessions of 300 seconds, with ten paired seed replications). The measurement results show that the FSA controller update cost was consistently 16–21 percent higher in the baseline pair and 9–15 percent higher in the extended pair, exhibiting a linear $O(N)$ growth pattern without evidence of superlinear degradation within the range of $N \leq 500$. Conversely, adding three new behaviours did not alter a single existing FSA module (0 lines modified), whereas the conventional controller required replacing the entire baseline controller file (335 lines; diff +188/−145) and increased the worst-case function complexity from 16 to 25. Furthermore, a lifecycle audit recorded 0 logic-animation divergences for the FSA, compared to 273.42 per 1,000 transitions for the conventional controller. These findings demonstrate that within the tested discrete state scope, the FSA guarantees code maintainability and animation transition determinism at the cost of a measurable and stable computational penalty.***

Keywords— finite state automata, state pattern, agent control logic, Three.js, game software architecture, cyclomatic complexity

Abstrak— **Logika kecerdasan buatan agen dan karakter non-pemain dalam *game* berbasis *web* harus berbagi sumber daya komputasi dengan proses *rendering* grafika, animasi *skeletal*, dan eksekusi perintah gambar WebGL. Paradigma *Finite State Automata* (FSA) dengan *State Pattern* dan logika kontrol imperatif konvensional sama-sama digunakan untuk mengorganisasi logika kontrol tersebut, namun pertukaran arsitektural dan komputasinya jarang diukur secara terkendali. Penelitian ini mengevaluasi *trade-off* kinerja komputasi, kualitas struktur kode, skalabilitas, dan determinisme perilaku kedua paradigma pada arsitektur *game* Three.js. Eksperimen terkendali memakai arena 3D dan stimulus persepsi bersama yang identik (jarak Euclidean dan Raycaster LoS); agen diformalkan sebagai $M = (Q, \Sigma, \delta, q_0)$ dengan lima perilaku diskret melalui *State Pattern* dengan *hook enter*()/*update*()/*exit*(), berbanding kontroler imperatif monolitik. Kinerja diukur melalui *timer* terisolasi yang membungkus murni fungsi pembaruan keputusan kontroler, di samping durasi skrip agregat, waktu *frame*, dan durasi fase *rendering*, disertai analisis statis AST Babel dan uji inferensial berpasangan pada rentang $N = 10–500$ agen (200 sesi 300 detik, sepuluh replikasi berpasangan *seed*). Hasil pengukuran menunjukkan bahwa biaya pembaruan kontroler FSA secara konsisten lebih tinggi 16–21 persen pada pasangan dasar dan 9–15 persen pada pasangan diperluas, dengan pola pertumbuhan linier $O(N)$ tanpa bukti degradasi superlinier pada rentang $N \leq 500$. Sebaliknya, penambahan tiga perilaku baru tidak mengubah satu pun modul FSA eksisting (0 baris dimodifikasi), sementara kontroler konvensional mengharuskan penggantian seluruh berkas kontroler dasar (335 baris; *diff* +188/−145) serta menaikkan kompleksitas fungsi terburuk dari 16 menjadi 25. Selain itu, audit siklus hidup mencatat 0 divergensi logika-animasi pada FSA berbanding 273,42 per 1.000 transisi pada kontroler konvensional. Temuan ini membuktikan bahwa pada cakupan status diskret yang diuji, FSA menjamin keterpeliharaan struktur kode dan determinisme transisi animasi dengan konsekuensi penalti komputasi yang terukur dan stabil.**

***Kata kunci*— finite state automata, state pattern, logika kontrol agen, Three.js, arsitektur perangkat lunak game, kompleksitas siklomatis**

I. PENDAHULUAN

Pengembangan logika kecerdasan buatan agen dan karakter non-pemain (*Non-Playable Character*/NPC) dalam lingkungan peramban web menghadapi batasan komputasi yang mendasar akibat arsitektur *single-threaded* pada *event loop* JavaScript [1]. Eksekusi logika agen harus berbagi siklus pemrosesan *Central Processing Unit* (CPU) dengan kalkulasi hierarki matriks grafika, pembaruan animasi skeletal, serta penerbitan perintah gambar menuju GPU melalui WebGL [2], [3]. Dalam kondisi tersebut, setiap milidetik yang dialokasikan untuk evaluasi keputusan agen merupakan milidetik yang tidak tersedia bagi penyusunan frame grafis, sehingga struktur kode pengambilan keputusan bukan sekadar pertimbangan estetika perangkat lunak melainkan faktor yang menentukan kelancaran pengalaman pengguna.

Dua paradigma rekayasa perangkat lunak mendominasi praktik pengelolaan logika kontrol agen 3D. Paradigma pertama memformalkan perilaku agen sebagai FSA dengan penerapan *State Pattern* berorientasi objek, di mana setiap status diisolasi ke dalam kelas mandiri yang mengimplementasikan antarmuka seragam dengan metode siklus hidup *enter*(), *update*(delta), dan exit() [4], [5]. Pada naskah ini, istilah FSA merujuk model formal perilaku diskret, *State Pattern* pada realisasi berorientasi objeknya, dan Skenario A pada pasangan keduanya. Paradigma kedua membangun logika kontrol tanpa abstraksi status formal, menyematkan pengambilan keputusan secara monolitik di dalam prosedur pembaruan utama menggunakan percabangan bersarang, dengan alur kontrol bergantung pada evaluasi serentak kombinasi variabel keadaan skalar maupun *boolean* [6].

Pemilihan di antara kedua paradigma ini umumnya didasarkan pada pertimbangan kualitatif dan intuisi rekayasa. Pada skala purwarupa dengan jumlah agen terbatas, pendekatan konvensional menawarkan kemudahan implementasi karena tidak memerlukan perancangan pola kelas maupun hierarki abstraksi objek, serta instruksi percabangan imperatif primitif dapat dioptimalkan secara agresif oleh kompilator *Just-In-Time* (JIT) melalui *inlining* jalur monomorfik [7]. Secara komputasi, evaluasi pohon percabangan bersarang menuntut penelusuran kondisi bertingkat berorde $O(\log k)$ hingga $O(k)$ predikat per agen untuk k perilaku, yang berpotensi memicu branch *misprediction* pada prosesor dan pemindaian bendera redundan seiring bertambahnya kompleksitas logika; sebaliknya, FSA mendelegasikan eksekusi langsung ke status aktif berorde $O(1)$. Namun, realisasi FSA berbasis objek memperkenalkan *overhead* pemanggilan metode dinamis (virtual dispatch) dan dereferensi pointer objek di memori *heap* yang dapat menghambat *inlining* JIT. Di sisi lain, klaim bahwa arsitektur modular memberikan keterpeliharaan yang lebih baik seiring pertumbuhan kompleksitas fitur [8], [9] jarang diverifikasi secara kuantitatif pada konteks spesifik arsitektur *game web*, terlebih dengan memperhatikan interaksinya terhadap subsistem animasi skeletal.

Beberapa jalur penelitian yang relevan telah berkembang. Studi arsitektur *game* mendokumentasikan penerapan mesin status untuk kecerdasan buatan agen [5], [10], [11], [12], termasuk implementasi FSM pada *game* 3D yang diteliti Bayu et al. [11], dan survei terkini merangkum teknik pengambilan keputusan karakter non-pemain beserta kecenderungan penelitiannya [13]. Penerapan kecerdasan buatan NPC pada *video game* 3D telah diimplementasikan lintas mesin *game* [14], sementara tinjauan sistematis perilaku NPC berbasis algoritma evolusioner [15] dan survei perencanaan dan pembelajaran dalam *game* [16] memetakan lanskap yang lebih luas, di samping pendekatan pohon keputusan pada *game* balap 3D [17]. Perbandingan terkontrol antar-representasi kebijakan juga telah dilakukan pada domain robotika untuk FSM berbanding *behavior tree* [18] dan pada *game* pertarungan untuk FSM berbanding pembelajaran penguatan mendalam [19]. Namun, studi-studi implementasi *game* tersebut umumnya berfokus pada kelayakan fungsional kualitatif pada *engine native* tanpa mengisolasi biaya *runtime* CPU per agen, sedangkan studi komparatif kebijakan belum menyentuh eksekusi pada *event loop* JavaScript *single-threaded* di peramban *web*. Di sisi lain, penelitian optimasi performa WebGL [3], [20] serta optimasi *game* [21] berfokus pada efisiensi *rendering* grafis dan *draw calls*, sementara beban komputasi logika agen diperlakukan sebagai konstanta yang tidak dianalisis. Studi metrik perangkat lunak menyediakan instrumen kompleksitas seperti kompleksitas siklomatis McCabe [8], metrik Halstead [9], evaluasi kognitif modern [22], dan *Maintainability Index* [23]; namun penerapannya pada basis kode *game* 3D masih terbatas. Kebaruan penelitian ini bukan terletak pada perbandingan konseptual FSA dan logika imperatif per se, melainkan pada rancangan eksperimen kuantitatif terisolasi yang memisahkan biaya CPU logika kontroler dari persepsi Raycaster dan *rendering* Three.js pada pipeline identik, pembuktian empiris fenomena divergensi logika-animasi pada AnimationMixer skeletal, serta kuantifikasi audit *Open–Closed Principle* (OCP) atas penambahan perilaku baru secara berpasangan pada *seed* acak yang sama.

Tujuan dan kontribusi penelitian ini dirumuskan secara tegas menguji pengaruh variabel independen arsitektur kontrol (FSA *State Pattern* vs Percabangan Imperatif Konvensional), varian kompleksitas fitur (dasar 5 perilaku vs diperluas 8 perilaku), dan skala populasi (N = 10, 50, 100, 250, 500 agen) terhadap variabel dependen kinerja *runtime* (biaya kontroler terisolasi, durasi skrip agregat, waktu *frame*, durasi *rendering*), kualitas struktur kode (McCabe, Halstead, MI, diff baris), dan integritas transisi perilaku (divergensi logika-animasi). Kontribusi penelitian ini ada tiga: Pertama, kerangka eksperimental terkendali dengan infrastruktur bersama yang menjamin kesetaraan perlakuan stimulus, lintasan, dan dinamika gerak agen antarskenario. Kedua, metodologi pemisahan metrik waktu ke dalam empat seri independen yang mengisolasi biaya murni pengambilan keputusan agen dari

overhead rendering dan fisika Three.js. Ketiga, pembuktian empiris *trade-off* arsitektural: kuantifikasi pertumbuhan kompleksitas dan biaya refaktorisasi penambahan fitur baru yang diaudit secara programatik, berbanding konsekuensi penalti waktu komputasi pembaruan kontroler pada mesin JavaScript.

II. METODE PENELITIAN

A. Desain Eksperimen Terkendali

Penelitian ini menerapkan pendekatan kuantitatif eksperimental dengan dua implementasi arsitektur kontrol yang dibandingkan dengan infrastruktur bersama; infrastruktur dan stimulus eksternal dipertahankan identik; komponen implementasi kontroler merupakan bagian dari perlakuan. Lingkup penelitian adalah siklus perilaku agen 3D pada peramban dengan lima status diskret: berdiam diri (*Idle*), berkeliling (*Patrol*), mengejar target (*Chase*), menyerang jarak dekat (*Attack*), dan melarikan diri (*Flee*). Untuk memenuhi syarat validitas internal, infrastruktur simulasi dipisahkan tegas dari lapisan keputusan: arena 3D, entitas agen, lintasan patroli, generator bilangan acak semu (*Pseudorandom Number Generator*/PRNG), penjadwal frame, subsistem animasi, resolusi tabrakan, dan seluruh parameter numerik berada pada modul bersama yang digunakan verbatim oleh kedua skenario, sehingga perbedaan terkurung sepenuhnya di dalam satu titik masuk *update*(*agent, delta*) pada kontroler masing-masing.

B. Formalisasi Automaton dan Skenario Pembanding

Pada Skenario A, agen diformalkan sebagai sistem transisi deterministik (kuadrupel):

$$M = (Q, \Sigma, \delta, q_0) \quad (1)$$

dengan himpunan status hingga:

$$Q = \{q_{idle}, q_{patrol}, q_{chase}, q_{attack}, q_{flee}\} \quad (2)$$

$q_0 = q_{idle}$ sebagai status awal. Status istirahat q_{idle} dan q_{patrol} ditandai sebagai keadaan stabil perilaku, bukan sebagai himpunan keadaan akhir penerima pada pengertian automaton klasik, karena agen dievaluasi terus-menerus selama simulasi berjalan; itulah alasan formalisme yang dipakai adalah sistem transisi tanpa komponen F. Sigma merupakan alfabet stimulus diskret yang diturunkan dari besaran lingkungan kontinu: keanggotaan ambang jarak serang dan radius lepas, keterlihatan garis pandang beserta memori target, keadaan kritis vitalitas, kesiapan serangan, serta simbol *fixture* (permintaan tameng, kehabisan amunisi, tertegun). $\delta: Q \times \Sigma \to Q$ bersifat deterministik dalam arti pasangan (status aktif, stimulus) yang sama selalu menghasilkan penerus tunggal; rantai transisi dalam satu *frame* dievaluasi berulang hingga konvergen pada status yang tidak lagi meminta penerus, dan pemeriksaan verifikasi menjamin rantai tersebut tidak bersiklus dalam satu *frame*.

Implementasi Skenario A merupakan komposit arsitektural yang memadukan beberapa pola desain yang berbeda perannya: (1) model FSA formal menyediakan himpunan status Q dan relasi transisi deterministik delta; (2) *State Pattern* berorientasi objek mengisolasi masing-masing status ke dalam kelas mandiri dengan hook siklus hidup *enter*(), *update*(delta), dan *exit*() sehingga logika di luar status aktif tidak pernah dievaluasi; (3) *Centralized Transition Guard* mengevaluasi stimulus interupsi prioritas tinggi secara terpusat sebelum status aktif mengevaluasi penerusnya sendiri; dan (4) *Flyweight Pattern* menginstansiasi setiap kelas status tepat satu kali dan membaginya ke seluruh entitas agen guna meniadakan alokasi memori *heap* berulang per transisi yang memicu jeda *garbage collection* pada mesin V8. Pemisahan dekomposisi ini krusial: manfaat modularitas dan keterpeliharaan berakar pada *State Pattern*, efisiensi alokasi memori berasal dari *Flyweight*, sedangkan penalti waktu komputasi berasal dari kombinasi pemanggilan metode terdelegasi (*virtual method dispatch*) dan evaluasi guard terpusat.

Skenario B dibangun tanpa abstraksi status formal. Pengambilan keputusan disematkan monolitik di dalam prosedur pembaruan utama menggunakan percabangan bersarang atas bendera keadaan seperti *isPatrolling*, *isChasing*, *isAttacking*, *currentHealth*, dan *targetDistance*; pada setiap *frame* fungsi utama memindai menyeluruh variabel tersebut untuk menentukan transformasi geometris dan pemicuan klip gerak. Skenario B bukan pembanding yang sengaja dilemahkan: kedua kontroler mengimplementasikan himpunan perilaku, ambang batas numerik, kecepatan lokomosi, dan antarmuka animasi yang sama, dan keduanya dievaluasi tepat satu kali per *frame*. Perbedaan jumlah transisi dan insiden divergensi bukan akibat rekayasa kecacatan, melainkan konsekuensi intrinsik dari perbedaan mekanisme kontrol: kontroler konvensional mengandalkan polling imperatif di setiap *frame* tanpa hook transisi status yang terikat siklus hidup klip animasi, sehingga rentan mengalami osilasi keputusan pada batas ambang sensorik; sedangkan FSA mempertahankan status aktif hingga transisi valid terpenuhi. Tingkat divergensi dihitung secara operasional melalui formula: Divergensi = (N_divergensi / N_transisi) x 1.000, di mana insiden divergensi tercatat ketika logika kontroler memutuskan perpindahan aksi (misalnya *Chase* ke *Attack*), namun THREE.AnimationMixer masih dalam fase pemudaran silang klip sebelumnya (crossFadeTo, T_blend = 0,25 s) sehingga *flag* isTransitioning memblokir pemicuan klip baru, menyebabkan status logika mendahului animasi yang sedang dimainkan. Algoritma 1 dan Algoritma 2 menyajikan *pseudocode* ringkas kedua kontroler; ambang numerik dan aturan stimulus lengkap tercantum pada Tabel I dan teks.

Algoritma 1: Pembaruan Kontroler FSA dalam Satu *Frame*
Masukan: status aktif q, stimulus Σ (jarak, vitalitas, LoS, amunisi), konteks agen a , δ_t ; Σ dihitung pada fase persepsi bersama di luar kontroler

1: $s \leftarrow Guard(\Sigma)$ // prioritas: tertegun > kritis > amunisi > tameng
2: jika s terdefinisi dan $s \neq q$ maka $q.exit(a)$; $s.enter(a)$; $q \leftarrow s$

3: ulangi sampai konvergen (tanpa siklus dalam satu frame):
    4: $q' \leftarrow q.evaluate(\Sigma)$ // hanya status aktif yang mengevaluasi
    5: jika $q' = q$ atau q' tidak terdefinisi maka berhenti
    6: $q.exit(a)$; $q'.enter(a)$; $q \leftarrow q'$ // hook berpasangan deterministik
7: $q.update(a, \delta_t)$ // logika di luar q tidak pernah dievaluasi
8: klip aktif = $klip(q)$ // enter() memicu tepat satu pemudaran silang

Algoritma 2: Pembaruan Kontroler Konvensional dalam Satu Frame
Masukan: bendera keadaan, stimulus Σ(jarak, vitalitas, LoS, amunisi); Σ dihitung pada fase persepsi bersama di luar kontroler

1: Σ (jarak, vitalitas, LoS, amunisi) tersedia dari fase persepsi bersama; tidak ada operasi persepsi di dalam kontroler
2: jika jeda patroli aktif dan tanpa ancaman maka bendera←IDLE; diam
3: jika vitalitas kritis maka
    4: jika target dekat maka bendera← FLEE; lakukanKabur
5: lain jika terlihat maka
    6: jika dalam jangkauan serang maka
        7: jika siap serang maka bendera←ATTACK; lakukanSerang
        8: lain jika sedang transisi animasi maka catat divergensi
        9: lain bendera←ATTACK; putar klip; tandai transisi berjalan
    10: lain bendera←CHASE; lakukanKejar(target terakhir diketahui)
11: lain bendera←PATROL; lakukanPatroli
12: rekonsiliasi: pastikan tepat satu bendera aktif // pasca-eksekusi

TABEL I
PEMETAAN PEMICU, OPERASI PERSEPSI, DAN PEMICUAN ANIMASI PADA KEDUA KONTROLER

Perilaku	**Pemicu (stimulus)**	**FSA (Skenario A)**	**Konvensional (Skenario B)**
Idle Patrol Chase	jeda titik patroli aktif; tanpa ancaman jeda habis; tanpa target terlihat target terlihat di luar jangkauan serang	q_{idle} menunggu jeda habis q_{patrol}: maju sepanjang jalur q_{chase}: kejar target	cabang bendera IDLE; diam cabang akhir pohon: patroli cabang terlihat: kejar; target terakhir diketahui bila tersembunyi
Attack	target dalam jangkauan dan siap serang	enter() memicu klip tepat sekali; damage saat *cooldown* habis	cabang dalam jangkauan; pemicu klip dijaga isTransitioning
Flee	vitalitas kritis dan target dekat	q_{flee}: menjauh dari ancaman	cabang kritis: menjauh

Tabel I memperlihatkan kesetaraan perlakuan: pemicu, ambang, kecepatan, dan antarmuka animasi identik pada kedua kontroler, dan seluruh persepsi (jarak *Euclidean*, vitalitas, serta uji keterlihatan LoS berbasis Raycaster) dihitung pada fase bersama sebelum kontroler dieksekusi, di luar *timer*, dengan kebijakan identik. Kedua kontroler menerima stimulus yang sama setiap frame; konsekuensinya, biaya raycast tidak termasuk dalam metrik biaya pembaruan kontroler dan perbedaan yang terukur mencerminkan organisasi logika kontrol pada dua implementasi spesifik yang diuji dengan stimulus eksternal identik; hasil ini tidak otomatis menggeneralisasi ke sebarang implementasi FSA atau kontroler konvensional.

C. Integrasi Animasi dan Persepsi

Dalam riset ini manajemen animasi menggunakan THREE.AnimationMixer yang mengoordinasikan AnimationClip melalui AnimationAction, dengan transisi antar klip dieksekusi melalui pemudaran silang:

$$action_{current}.crossFadeTo(action_{next}, T_{blend}, warp) \quad (3)$$

Parameter *warp* menginterpolasi skala waktu kedua klip secara proporsional sehingga meniadakan pergeseran langkah kaki saat agen beralih kecepatan. Mekanisme ini sengaja dibagi identik oleh kedua paradigma karena eksperimen membandingkan kapan transisi diputuskan, bukan bagaimana dieksekusi. Persepsi mengombinasikan jarak Euclidean tiga dimensi dengan uji keterlihatan THREE.Raycaster; okulasi garis pandang oleh halangan interior menjadikan raycasting sebagai komputasi yang nyata; hasilnya dihitung pada fase persepsi bersama dan biayanya berada di luar timer kontroler. Pergerakan menuju dan menjauhi target mengikuti perilaku kemudi *seek* dan *flee* yang lazim pada pergerakan karakter non-pemain [24], dan rotasi orientasi menggunakan interpolasi kuaternion bola

$$q_{interpolated} = slerp(q_{current}, q_{target}, \alpha)$$

dengan $\alpha = min(1, \omega \times \delta_t)$.

D. Prosedur Pengukuran

Prosedur terdiri atas enam tahap sebagaimana diperlihatkan pada Gambar 1 Pembangunan arena, formalisasi automaton, pembangunan skenario pembanding, integrasi animasi, instrumentasi, dan pengujian berjenjang pada N = 10, 50, 100, 250, dan 500 agen aktif. *Loop* simulasi mencatat empat seri waktu secara independen: *frame time* (seluruh langkah termasuk renderer.render()), durasi skrip CPU agregat (fase kontrol, *mixer* animasi, integrasi transformasi matriks), biaya pembaruan kontroler (terisolasi: hanya pemanggilan agent.controller.update()), dan durasi *fase rendering* (submisi perintah gambar WebGL *draw calls*). *Timer* seri ketiga membungkus murni fungsi pembaruan kontroler: evaluasi penjaga dan kondisi transisi, keputusan aksi, serta penugasan kecepatan; pembaruan stimulus persepsi bersama, integrasi transformasi posisi, *mixer* animasi, dan *rendering* berada di luar *timer*. Uji keterlihatan Raycaster berada pada fase persepsi bersama di luar timer sehingga biayanya tidak mencemari metrik kontroler; *timer* mengisolasi biaya pembaruan kontroler dari persepsi, animasi, dan *rendering*. Pemisahan seri ketiga bersifat krusial: biaya pembaruan kontroler menyumbang sekitar 7–8 persen durasi skrip agregat pada N = 500 (FSA 1,1602/14,259 = 8,1 persen; konvensional 0,9916/14,098 = 7,0 persen), sehingga pengujian yang semata-mata menggunakan metrik agregat akan menguji infrastruktur pengukuran bersama, bukan karakteristik logika kontrol. Sebagai kontrol negatif, durasi fase *rendering* tidak berbeda secara sistematis karena jalur rendering dirancang identik untuk kedua skenario.



Gambar 1. Diagram alir prosedur penelitian: enam tahap dari pembangunan arena hingga analisis statistik

Setiap sesi pengukuran didahului pemanasan 2 detik yang dibuang agar perilaku kompilator JIT tidak dibebankan kepada arsitektur, dijalankan pada halaman peramban baru agar tumpukan memori tidak diwarisi, dan menggunakan pembibitan acak semu (PRNG) identik per replikasi sehingga perbandingan bersifat berpasangan. Protokol penuh terdiri atas 200 sesi berdurasi 300 detik: empat konfigurasi (dua paradigma, masing-masing dalam varian dasar dan diperluas tiga fitur perilaku baru), lima tingkat populasi, dan sepuluh replikasi berpasangan (*seed* 1001–1010). Eksekusi dilakukan pada server *virtual* Ubuntu 22.04 LTS x86-64 dengan 8 vCPU (Intel Xeon @ 2.50 GHz, AVX-512), 8 GB RAM, Node.js v20.15, Three.js r163, dan Chromium 127 headless dengan flag: --headless=new, --disable-gpu, --use-gl=angle, --use-angle=swiftshader, --disable-dev-shm-usage, viewport 1280x720, dan vsync dinonaktifkan. Penggunaan rasterisasi perangkat lunak (SwiftShader) merupakan keputusan metodologis yang disengaja untuk mengeliminasi variabilitas penjadwalan driver GPU, sinkronisasi VSync, dan thermal throttling grafis, sehingga perbedaan siklus pemrosesan CPU murni terisolasi. Sesi dieksekusi berselang-seling untuk mendistribusikan efek urutan secara merata. Instrumentasi waktu menggunakan API presisi tinggi W3C *performance.now*() pada loop simulasi di luar kedua kontroler, sehingga biaya pengukuran identik.

E. Teknik Analisis Data

Analisis statis menggunakan parsing AST Babel atas berkas sumber logika kontrol tiap skenario, menghitung kompleksitas siklomatis McCabe:

$$M = E - N + 2P \qquad (4)$$

yang untuk program terstruktur setara satu ditambah jumlah titik keputusan; volume Halstead $V = N \cdot \log_2(\eta)$ dengan $\eta = \eta_1 + \eta_2$ dan $N = N_1 + N_2$; tingkat kesulitan $D = \left(\frac{\eta_1}{2}\right) \times \left(\frac{N_2}{\eta_2}\right)$; upaya $E = D \times V$; serta *Maintainability Index*:

$$MI = \max\left(0, \frac{100 \times (171 - 5.2 \ln V - 0.23M - 16.2 \ln LOC)}{171}\right) \qquad (5)$$

MI dilaporkan sebagai distribusi per modul karena ia merupakan properti per modul; penjumlahan volume dan LOC lebih dulu akan mendorong setiap unit multi-berkas menuju nol dan menghancurkan perbandingan. Total agregat juga tidak dipakai sebagai pembanding utama karena tidak adil membandingkan arsitektur banyak modul kecil dengan satu prosedur besar; angka yang diskriminatif adalah kompleksitas modul terburuk dan fungsi terburuk. Varian MI yang digunakan adalah model polinomial Visual Studio pada skala 0–100 dengan unit analisis berkas sumber; nilai MI rata-rata adalah rata-rata antar berkas pada unit perlakuan. Aturan Halstead mengikuti definisi klasik: operator mencakup operator aritmetika, logika, relasional, dan penugasan, kata kunci alur kendali, serta akses anggota; operan mencakup identifier dan literal. Penghitungan baris diperoleh dari diff objektif antara berkas kontroler dasar dan varian diperluas: +188 baris ditambahkan dan 145 baris dihapus dari 335 baris berkas dasar; pada FSA, modul *baseline* tidak berubah (SHA-256 identik). Artefak diff tersedia pada suplemen.

Analisis runtime menggunakan statistika inferensial: uji Shapiro-Wilk untuk normalitas:

$$W = \frac{\left(\sum_i a_i x_{(i)}\right)^2}{\left(\sum_i x_i - \underline{x}\right)^2} \qquad (6)$$

uji-t berpasangan, dengan $D_i = X_{1i} - X_{2i}$ sebagai selisih pasangan dan t = rata-rata D dibagi galat baku D:

$$t = \frac{\bar{D}}{s_D/\sqrt{n}} \qquad (7)$$

dengan *fallback* nonparametrik berpasangan Wilcoxon signed-rank apabila normalitas selisih ditolak; d_z (selisih rata-rata dibagi simpangan baku selisih) untuk besaran efek; analisis varians dua arah:

$$Y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ijk} \qquad (8)$$

untuk efek utama arsitektur, populasi, dan interaksinya; serta regresi kuadratik untuk pola pertumbuhan:

$$t_{script}(N) = \beta_0 + \beta_1 N + \beta_2 N^2 + \varepsilon \qquad (9)$$

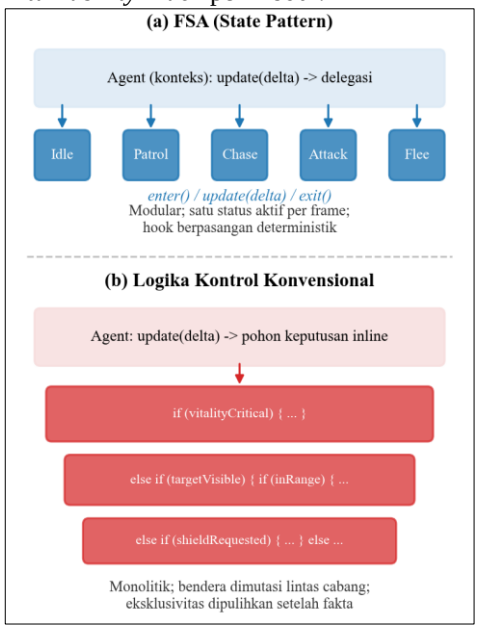
dengan R^2 dan uji signifikansi parsial atas koefisien kuadratik β_2 dievaluasi untuk menguji hipotesis degradasi superlinier. Struktur hierarki data adalah: 1 replikasi = 1 pasangan seed identik; 1 sesi 300 s (~18.000 *frame*) direduksi menjadi 1 statistik ringkasan per sesi; unit analisis inferensial adalah rata-rata sesi (n = 10 observasi independen per sel perlakuan), menghindari bias pseudoreplikasi *frame*. Karena replikasi ke-k kedua skenario menerima seed dunia identik, uji-t berpasangan dilaporkan dengan koreksi Holm-Bonferroni atas lima tingkat beban populasi (alpha = 0,05). *Fallback* nonparametrik adalah Wilcoxon *signed-rank* yang mempertahankan pasangan; kesimpulan utama bersandar pada uji berpasangan yang sesuai desain

III. HASIL DAN PEMBAHASAN

A. Kontras Struktural dan Analisis Statis

Gambar 2 memperlihatkan kontras struktural yang menjadi objek pengukuran. Pada Skenario A, entitas agen mendelegasikan pembaruan frame kepada tepat satu instans status aktif sehingga *hook enter*() dan *exit*() berpasangan deterministik; pada Skenario B seluruh pohon keputusan dievaluasi dalam satu prosedur dan eksklusivitas bendera dipulihkan secara pasca-eksekusi melalui rekonsiliasi di akhir *frame*.

Tabel II menyajikan metrik kode sumber baik pada tingkat agregat unit perlakuan maupun rincian per berkas sumber yang dianalisis langsung melalui AST Babel. Perbedaan paling mencolok terletak pada kompleksitas fungsi terburuk: pada fsa-controller.js fungsi terburuk hanya bernilai 7 (fungsi evaluasi transisi) dan pada states.js bernilai 6, sedangkan pada kontroler konvensional fungsi pembaruan utama mencapai 16 pada konfigurasi dasar dan melonjak hingga 25 pada konfigurasi diperluas. Total agregat kompleksitas siklomatis memang lebih tinggi pada FSA (86 vs 52 pada dasar; 130 vs 71 pada diperluas), namun angka agregat bertambah secara mekanis seiring jumlah berkas dan kelas sehingga bukan indikator keterpeliharaan yang adil; indikator diskriminatif yang mencerminkan beban kognitif pemeliharaan kode adalah kompleksitas fungsi terburuk dan *Maintainability Index* per modul.



Gambar 2. Diagram alir prosedur penelitian: enam tahap dari pembangunan arena hingga analisis statistik

TABEL III
KARAKTERISTIK KODE SUMBER DAN METRIK STATIS PER UNIT PERLAKUAN DAN RINCIAN PER BERKAS SUMBER (BABEL AST)

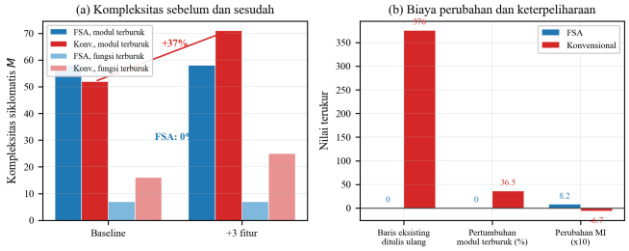
Unit / Berkas Modul	Peran Arsitektural	LOC Logis	M Total	M Fungsi Terburuk	Volume Halstead	MI
[Unit] FSA (dasar)	Agregat 2 modul baseline	297	86	7	8.771,6	21,8
[Unit] Konvensional (dasar)	Prosedur monolitik baseline	210	52	16	7.009,2	15,4
[Unit] FSA (diperluas)	Agregat 4 modul diperluas	464	130	7	13.702,8	30,0
[Unit] Konvensional (diperluas)	Prosedur monolitik diperluas	291	71	25	9.980,4	8,7
src/fsa/states.js	Kelas status dasar & transisi	167	58	6	4.778,3	18,0
src/fsa/fsa-controller.js	Mesin FSA & guard terpusat	130	28	7	3.155,8	25,6
src/conventional/conventional-controller.js	Pohon keputusan dasar (5 status)	210	52	16	7.009,2	15,4
src/variants/fsa-scaled-states.js	Modul status tambahan (3 status)	145	41	6	3.736,6	22,3
src/variants/fsa-scaled-controller.js	Adapter registrasi status baru	22	3	1	207,5	54,1
src/variants/conventional-scaled-controller.js	Pohon keputusan diperluas (8 status)	291	71	25	9.980,4	8,7

Unit	M sum	M modul terburuk	M fungsi terburuk	LOC logis	Volume Halstead	MI rata-rata
FSA (dasar)	86	58	7	297	8.771,60	21,8
Konvensional (dasar)	52	52	16	210	7.009,20	15,4
FSA (diperluas)	130	58	7	464	13.702,80	30
Konvensional (diperluas)	71	71	25	291	9.980,40	8,7

TABEL IIIII
PERUBAHAN METRIK KODE SETELAH PENAMBAHAN TIGA FITUR PERILAKU

Indikator	FSA	Konvensional
Kompleksitas siklomatis maksimum per modul, sebelum → sesudah	58 → 58 (0%)	52 → 71 (+36,5%)
Kompleksitas siklomatis maksimum per fungsi, sebelum → sesudah	7 → 7	16 → 25
MI rata-rata, sebelum → sesudah	21,8 → 30,0 (+8,2)	15,4 → 8,7 (−6,7)
Modul status/kontroler baru yang ditambahkan	1	0
Baris kode lama yang berubah	0	+188 / −145 (diff)
Baris pada berkas kontroler dasar yang digantikan oleh varian diperluas	—	335

Tabel III melaporkan temuan paling substantif analisis statis. Tiga fitur perilaku baru (*Defend*, SearchAmmo, *Stunned*) ditambahkan ke kedua basis kode. Pada FSA, ketiganya direalisasikan sebagai tiga kelas status mandiri pada satu modul baru, tanpa mengubah satu baris kode eksisting dan tanpa menaikkan kompleksitas modul terburuk; *Maintainability Index* rata-rata justru naik dari 21,8 menjadi 30,0 karena modul baru lebih sederhana daripada rata-rata modul eksisting. Pada konvensional, fitur yang sama menaikkan kompleksitas modul terburuk 36,5 persen (52 menjadi 71), fungsi terburuk dari 16 menjadi 25, mengharuskan penggantian seluruh berkas kontroler dasar (335 baris; diff +188/−145) eksisting ditulis ulang, dan menurunkan *Maintainability Index* dari 15,4 menjadi 8,7. Mekanisme yang mendasarinya adalah sebagai berikut: karena setiap status hanya mengembalikan label tujuan dan mesin menyelesaikannya terhadap kumpulan instans, label baru tidak mengharuskan modifikasi kelas lama; keterjangkauannya disediakan satu fungsi penjaga terpusat yang dievaluasi sebelum status aktif, sehingga setiap fitur baru memerlukan tepat satu aturan penjaga pada satu tempat. Gambar 3 memvisualisasikan biaya perubahan tersebut.



Gambar 3. Konsekuensi struktural penambahan tiga fitur: (a) kompleksitas modul dan fungsi terburuk sebelum dan sesudah; (b) biaya perubahan dalam baris ditulis ulang, pertumbuhan modul terburuk, dan perubahan Maintainability Index.

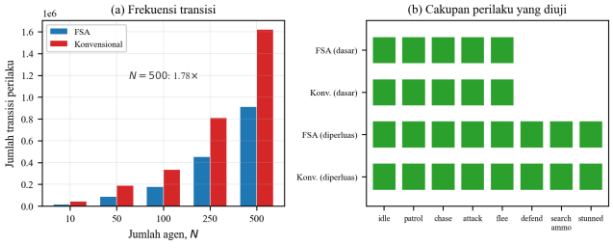
B. Verifikasi Determinisme dan Integritas Automaton

Audit siklus hidup dijalankan programatik terhadap setiap agen pada setiap sesi konfigurasi FSA. Karena jumlah agen per replikasi adalah 10+50+100+250+500 = 910 dan terdapat sepuluh replikasi, tiap automaton memperoleh 9.100 audit sesi-agen; varian dasar dan varian diperluas masing-masing diaudit terpisah (total 18.200), dan seluruhnya seimbang. Audit memverifikasi invarian bahwa untuk setiap status q berlaku enters(q) - exits(q) = 1 apabila q aktif dan nol apabila bukan. Verifikasi konvergensi rantai transisi dalam satu frame, yaitu bahwa rantai evaluasi berulang di dalam satu frame tidak bersiklus, dilakukan menyeluruh terhadap 1.920 kombinasi pasangan status-stimulus pada automaton dasar dan 12.288 kombinasi pada automaton diperluas, dengan hasil nol transisi ilegal pada keduanya. Verifikasi ini menemukan defek nyata yang tidak terdeteksi inspeksi: *deadlock* pada batas ambang ketika tiga status menyandikan definisi ancaman yang berbeda (*chase* - *flee* - *patrol* - *chase* dalam satu *frame*, 120 kasus terjebak), serta dua status yang awalnya tidak terjangkau. Perbaikannya memusatkan definisi ancaman ke satu predikat bersama sehingga relasi transisi konsisten secara konstruksi. Temuan ini menunjukkan bahwa determinisme bukan properti otomatis paradigma formal melainkan hasil disiplin verifikasi; keunggulan FSA pada implementasi ini adalah tersedianya kerangka verifikasi programatik atas invarian transisi dan siklus hidup.

TABEL IVV
VERIFIKASI DETERMINISME DAN INTEGRITAS SIKLUS HIDUP AUTOMATON

Indikator	FSA	Konvensional
Audit sesi-agen seimbang (dasar / diperluas / total)	9.100/9.100; 9.100/9.100; 18.200/18.200	tidak berlaku
Transisi ilegal	0	0
Kombinasi divalidasi (dasar / diperluas)	1.920 / 12.288	-
Status tidak terjangkau (dasar / diperluas)	0 dari 5 / 0 dari 8	-
Divergensi logika-animasi per 1.000 transisi	0	273,42
Transisi relatif (konvensional / FSA)	-	1,82x

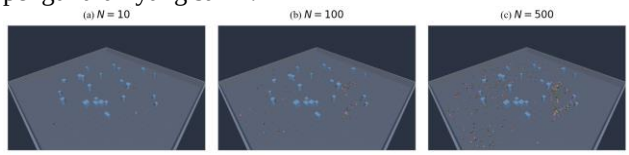
Tabel IV dan Gambar 4 merangkum aspek frekuensi transisi dan cakupan perilaku. Kontroler konvensional mengubah keputusannya 1,82 kali lebih sering di bawah aliran stimulus identik, yang mencakup masuk-kembali sah pada batas ambang persepsi dan bukan indikator kualitas perilaku itu sendiri, serta mencatat 273,42 insiden divergensi logika-animasi per 1.000 transisi. Satu insiden divergensi didefinisikan sebagai satu episode ketika permintaan transisi ditekan karena pemudaran klip sebelumnya belum selesai sehingga status logika berbeda dari klip yang diputar; penyebutnya adalah jumlah transisi tercatat. Pada implementasi ini FSA tidak mencatat satu pun divergensi; mekanisme spesifiknya adalah *enter*() yang dipanggil tepat satu kali per transisi, sehingga tidak ada jalur eksekusi yang dapat menahan permintaan transisi selama pemudaran silang masih berlangsung. Ini merupakan temuan pada implementasi yang diuji, bukan jaminan otomatis bagi sebarang implementasi FSA. Transisi ilegal, yang bernol pada kedua pendekatan, didefinisikan bagi kontroler konvensional sebagai keadaan pasca-rekonsilasi dengan jumlah bendera perilaku aktif tidak tepat satu. Cakupan perilaku pada sesi 300 detik bersifat penuh: 5 dari 5 status dasar di setiap sesi dan 8 dari 8 status diperluas di setiap sesi, sehingga perbandingan determinisme tidak kosong pada bagian ruang status mana pun.



Gambar 4. Determinisme dan cakupan perilaku: (a) frekuensi transisi kedua paradigma; (b) matriks cakupan perilaku yang membuktikan ruang status mana yang benar-benar diuji tiap skenario.

C. Hasil Pengukuran Runtime Protokol Penuh

Gambar 5 memperlihatkan arena simulasi pada tiga tingkat beban sebagai konteks visual; kepadatan pada N = 500 memperlihatkan mengapa pemisahan beban CPU logika kontrol dari beban rendering menjadi prasyarat pengukuran yang sahih.



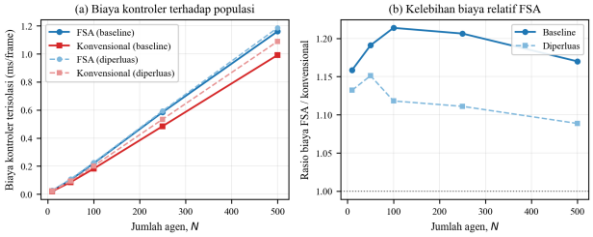
Gambar 5. Arena simulasi pada tiga tingkat beban populasi: tangkapan layar langsung dari aplikasi berjalan (skenario FSA, seed 1337); agen divisualisasikan delapan palet warna dan pemain ditandai beacon vertikal.

TABEL V
BIAYA PEMBARUAN KONTROLER PER FRAME (MS/FRAME), RATA-RATA ± SD 10 REPLIKASI; P = UJI-T BERPASANGAN DENGAN KOREKSI HOLM; ATAS: PASANGAN DASAR, BAWAH: PASANGAN DIPERLUAS

Metrik	Varian	FSA	Konvensional	Rasio	p
Biaya pembaruan kontroler, N = 10	Dasar	0,0247 ± 0,0013	0,0213 ± 0,0015	1,16	$9{,}1 \times 10^{-6}$
Biaya pembaruan kontroler, N = 50	Dasar	0,1022 ± 0,0037	0,0858 ± 0,0031	1,19	$3{,}7 \times 10^{-6}$
Biaya pembaruan kontroler, N = 100	Dasar	0,2206 ± 0,0079	0,1818 ± 0,0051	1,21	$3{,}7 \times 10^{-6}$
Biaya pembaruan kontroler, N = 250	Dasar	0,5849 ± 0,0104	0,4848 ± 0,0094	1,21	$5{,}2 \times 10^{-10}$
Biaya pembaruan kontroler, N = 500	Dasar	1,1602 ± 0,0374	0,9916 ± 0,0257	1,17	$9{,}1 \times 10^{-6}$
Durasi skrip CPU agregat, N = 500 (ms)	Dasar	14,259	14,098	1,01	0,666
Durasi fase rendering, N = 500 (ms)	Dasar	41,228	41,498	0,99	0,610
Biaya pembaruan kontroler, N = 10	Diperluas	0,0270 ± 0,0020	0,0238 ± 0,0020	1,13	$9{,}0 \times 10^{-3}$
Biaya pembaruan kontroler, N = 50	Diperluas	0,1081 ± 0,0043	0,0939 ± 0,0040	1,15	$1{,}4 \times 10^{-4}$
Biaya pembaruan kontroler, N = 100	Diperluas	0,2250 ± 0,0048	0,2012 ± 0,0067	1,12	$2{,}0 \times 10^{-5}$
Biaya pembaruan kontroler, N = 250	Diperluas	0,5936 ± 0,0164	0,5343 ± 0,0171	1,11	$1{,}7 \times 10^{-4}$
Biaya pembaruan kontroler, N = 500	Diperluas	1,1855 ± 0,0416	1,0889 ± 0,0372	1,09	$3{,}9 \times 10^{-4}$

Tabel V melaporkan hasil utama pengukuran runtime pada empat metrik terpisah: biaya pembaruan kontroler (terisolasi), durasi skrip CPU agregat, durasi fase *rendering*, dan *frame time* total. Biaya pembaruan kontroler FSA secara konsisten lebih tinggi pada metrik primer dengan rasio 1,16–1,21 pada pasangan dasar, signifikan pada seluruh lima tingkat beban menurut uji-t berpasangan pada pasangan *seed* dengan koreksi Holm-Bonferroni (p terkecil 5,2 x 10^-10). Pada N = 500, selisih berpasangan adalah 0,1686 milidetik dengan simpangan baku 0,0591 milidetik dihitung atas selisih antar pasangan seed dan interval kepercayaan 95 persen 0,126–0,211 milidetik, dengan ukuran efek berpasangan d_z bernilai 3,07 [IK95: 1,36–4,78]; 3,71 [1,70–5,73]; 3,69 [1,69–5,69]; 10,46 [5,12–15,80]; dan 2,85 [1,24–4,46] untuk N = 10; 50; 100; 250; 500 berurutan. Nilai d_z yang sangat besar ini mencerminkan varians selisih pasangan *seed* yang sangat kecil (SD_D = 0,0591 ms pada N = 500) berkat lingkungan eksperimen terkontrol dan deterministik. Namun, relevansi praktisnya perlu ditafsirkan secara proporsional terhadap anggaran *frame* grafis: pada N = 500 agen, selisih waktu kontroler absolut hanya 0,1686 ms per *frame* (1,1602 ms vs 0,9916 ms). Dalam konteks anggaran *frame* standar 16,67 ms (target 60 FPS) atau 33,33 ms (target 30 FPS), selisih 0,17 ms ini hanya menyerap 1,01 persen dari anggaran 60 FPS dan 0,51 persen dari anggaran 30 FPS. Artinya, meskipun overhead pembaruan kontroler FSA secara statistik sangat signifikan dengan *effect size* masif, dampak praktisnya terhadap penurunan *frame rate* pada perangkat pengguna nyata dapat diabaikan untuk sebagian besar skenario game interaktif.

Pada pasangan konfigurasi diperluas, rasio biaya FSA per tingkat beban adalah 1,13; 1,15; 1,12; 1,11; dan 1,09 untuk N = 10 hingga 500, seluruhnya signifikan menurut uji-t berpasangan berkoreksi Holm dengan p = 9,0 x 10^-3; 1,4 x 10^-4; 2,0 x 10^-5; 1,7 x 10^-4; dan 3,9 x 10^-4 untuk N = 10 hingga 500. Pemeriksaan asumsi normalitas melalui Shapiro-Wilk atas selisih FSA-konvensional per seed tidak ditolak pada seluruh sepuluh sel perlakuan (dua pasangan konfigurasi kali lima beban; p terkecil 0,811), sehingga uji-t berpasangan berlaku pada setiap sel. Durasi skrip CPU agregat (fase kontroler ditambah *mixer* animasi dan pembaruan matriks Three.js) serta durasi fase *rendering* tidak berbeda signifikan antarskenario (p = 0,666 dan p = 0,610 pada N = 500), mengonfirmasi bahwa komponen infrastruktur grafis bersama mendominasi durasi total *frame* dan menegaskan pentingnya isolasi timer kontroler. Gambar 6 menampilkan pola penskalaannya: rasio overhead berada pada rentang stabil 16–21 persen terhadap populasi.



Gambar 6. Biaya pembaruan kontroler terhadap populasi untuk keempat konfigurasi (a) dan kelebihan biaya relatif FSA (b); setiap titik rata-rata 10 replikasi sesi 300 detik.

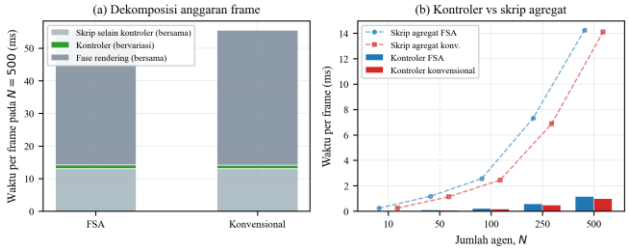
Tabel VI merangkum dekomposisi varians dan evaluasi model pertumbuhan. Karena ANOVA tidak memodelkan pasangan *seed* secara eksplisit, ia diposisikan sebagai analisis eksploratif-deskriptif; inferensi konfirmatori utama naskah ini adalah uji berpasangan berkoreksi Holm pada Tabel V. ANOVA atas pasangan dasar (100 nilai ringkasan sesi; df arsitektur 1, populasi 4, interaksi 4, galat 90) menunjukkan efek utama arsitektur yang besar pada analisis eksploratif (F = 450,96; eta kuadrat parsial 0,834), efek populasi dominan, serta interaksi signifikan (F = 98,94): rasio bervariasi ringan dalam rentang sempit. Pada durasi skrip agregat, efek arsitektur tidak signifikan (F = 3,30; p = 0,073), membuktikan secara empiris bahwa metrik agregat tidak sensitif terhadap perbedaan arsitektural kontroler.

Uji regresi kuadratik $t_{kontroler}(N) = \beta_0 + \beta_1 N + \beta_2 N^2 + \varepsilon$ mengonfirmasi bahwa pertumbuhan biaya kedua kontroler adalah murni linear terhadap populasi N. Koefisien kuadratik β_2 bernilai mendekati nol: $-3{,}09 \times 10^{-}8$ (p = 0,895) untuk FSA dan $+2{,}23 \times 10^{-}7$ (p = 0,262) untuk konvensional. Karena uji signifikansi parsial atas β_2 tidak signifikan (p > 0,05) pada kedua kontroler, penambahan suku kuadratik tidak memberikan peningkatan kecocokan yang bermakna dibanding model linear sederhana. Nilai R^2 yang tinggi (0,9997 dan 0,9998) mencerminkan kedekatan data terhadap garis tren, bukan bukti adanya kurvatur nonlinier, sehingga hipotesis degradasi superlinier pada kontroler konvensional ditolak pada rentang N ≤ 500.

TABEL VI
ANALISIS VARIANS DUA ARAH (EKSPLORATIF) DAN REGRESI PERTUMBUHAN BIAYA KONTROLER

Model	Sumber variasi	F	p	Eta kuadrat parsial
ANOVA dua arah, waktu kontroler	Arsitektur	450,96	< 0,001	0,834
	Populasi	15.776,25	< 0,001	0,999
	Interaksi	98,94	< 0,001	0,815
ANOVA dua arah, skrip agregat	Arsitektur	3,3	0,073	0,035
Regresi kuadratik, FSA	beta2	-3,09 x 10^-8	0,895	R^2 = 0,9997
Regresi kuadratik, konvensional	beta2	2,23 x 10^-7	0,262	R^2 = 0,9998

Gambar 7 menyajikan bukti visual dekomposisi anggaran frame yang menjadi dasar klaim metodologis tersebut.



Gambar 7. Dekomposisi anggaran frame pada N = 500 (a) dan kontras biaya pembaruan kontroler terhadap skrip agregat (b): selisih kontroler yang nyata dan signifikan tenggelam dalam komponen bersama bila hanya skrip agregat yang diukur.

D. Analisis Temuan

Penyebab utama tingginya biaya pembaruan kontroler FSA (16–21% lebih lambat) dapat ditelusuri ke karakteristik eksekusi mesin JavaScript V8 TurboFan. Kontroler konvensional dibangun di atas satu fungsi monolitik dengan percabangan imperatif searah. Kompilator JIT dapat dengan mudah melakukan inlining pada percabangan monomorfik ini, menjaga variabel keadaan skalar tetap berada di dalam register CPU tanpa overhead pembentukan stack frame baru. Sebaliknya, realisasi FSA menggunakan State Pattern mengandalkan pemanggilan metode polimorfik agent.currentState.update (agent, delta). Di setiap frame, mesin JavaScript harus melakukan pembacaan pointer objek ke heap memory dan penelusuran prototype chain, yang memicu polymorphic inline cache (PIC) misses ketika agen berada pada status yang beragam di dalam populasi. Selain itu, pendelegasian ke fungsi Guard(sigma) terpusat menambah kedalaman tumpukan pemanggilan (call stack depth). Fakta bahwa penalti ini relatif stabil (16–21%) menunjukkan bahwa biaya indirection objek ini merupakan konstanta overhead pemanggilan metode per entitas, bukan kegagalan algoritmik yang memburuk terhadap skala populasi.

Temuan ini secara tegas menunjukkan adanya trade-off rekayasa perangkat lunak, bukan keunggulan mutlak satu paradigma atas paradigma lainnya. Kontroler konvensional terbukti lebih efisien secara komputasi (16–21% lebih cepat, zero method indirection), menjadikannya pilihan optimal untuk purwarupa cepat, agen dengan alur keputusan sederhana dan statis, atau aplikasi web game yang menargetkan perangkat keras berspesifikasi rendah dengan anggaran CPU ketat. Sebaliknya, FSA *State Pattern* unggul dalam keterpeliharaan struktural dan integritas transisi perilaku, dengan membayar premi komputasi yang terukur dan stabil.

Hubungan antara metrik kompleksitas kode dan keterpeliharaan (*maintainability*) berakar pada prinsip desain perangkat lunak Solid, khususnya OCP dan *Single Responsibility Principle* (SRP). Pada kontroler konvensional, penambahan tiga perilaku baru meningkatkan kompleksitas siklomatis fungsi terburuk dari 16 menjadi 25. Berdasarkan taksonomi McCabe, fungsi dengan kompleksitas di atas 10 tergolong rawan cacat (*high risk*) dan sulit diuji secara memadai karena menuntut setidaknya 25 jalur pengujian independen. Selain itu, penggabungan seluruh evaluasi ke dalam satu berkas monolitik mengharuskan penulisan ulang seluruh logika kontroler dasar (335 baris digantikan dengan diff +188/−145 baris), yang secara signifikan meningkatkan risiko regresi (*bug regression*) pada jalur keputusan lama yang sebelumnya stabil. Sebaliknya, pada FSA State Pattern, setiap perilaku diisolasi ke dalam kelas mandiri dengan kompleksitas fungsi terburuk tetap stabil pada angka 7. Penambahan fitur baru direalisasikan melalui modul baru tanpa memodifikasi satu baris pun modul lama (OCP terpenuhi), sehingga pengujian unit dapat dilakukan secara terisolasi per kelas status.

Keterbatasan struktural FSA tetap harus diakui: ketika perilaku agen menjadi sangat kompleks dan melibatkan aksi simultan (misalnya menembak sambil bergerak), FSA klasik menghadapi ledakan status kombinatorial ($|Q| = |Q_{gerak}| \times |Q_{aksi}|$). Dalam konteks tersebut, arsitektur seperti Hierarchical FSM (HFSM) atau *Behavior Tree* (BT) [25], [26] lebih disarankan. Namun pada domain fase perilaku diskret yang saling lepas, cakupan penelitian ini, FSA memberikan jaminan determinisme siklus hidup yang dapat diverifikasi secara programatik.

E. Implikasi

Bagi pengembang game web yang mengantisipasi evolusi fitur perilaku, FSA berbasis *State Pattern* dengan penjaga transisi terpusat menghilangkan kebutuhan menulis ulang logika eksisting; biaya adopsinya adalah perancangan hierarki kelas status dan antarmuka siklus hidup. Logika kontrol konvensional tetap wajar dan efisien untuk purwarupa atau *game* dengan cakupan perilaku stabil. Dalam hal anggaran frame, pada perangkat pengguna nyata dengan target 60 FPS (anggaran 16,67 ms), *overhead* kontroler FSA sebesar 0,0388 ms pada N = 100 dan 0,1686 ms pada N = 500 hanya menyerap 0,23–1,01 persen dari total anggaran frame, membuktikan bahwa *overhead* ini tidak menjadi hambatan praktis pada perangkat modern. Kontribusi metodologis yang lebih umum adalah bahwa pemisahan metrik waktu ke dalam seri independen merupakan keharusan pada setiap eksperimen komparatif dengan proporsi kode bersama besar, karena metrik gabungan akan gagal mendeteksi perbedaan yang secara struktural ada

F. Keterbatasan

Rasterisasi perangkat lunak menyebabkan durasi frame absolut melampaui anggaran 16,67 milidetik (sekitar 56 milidetik per frame pada N = 500); perbandingan CPU relatif tetap sahih karena kedua skenario menempuh jalur rendering sama dan pemeriksaan pemalsuan terpenuhi, namun klaim anggaran *frame* pada perangkat GPU tidak dapat ditarik. Ketiadaan bukti komponen kuadratik hanya berlaku pada rentang N≤500 yang diuji. Penelitian menggunakan satu domain kasus (arena agen pengejar pada peramban); generalisasi ke domain arsitektur game lain belum diverifikasi. Replikasi pada perangkat GPU, beban lebih tinggi, dan perbandingan dengan *Behavior Tree* disarankan sebagai penelitian lanjutan.

IV. KESIMPULAN

Penelitian ini mengevaluasi perbandingan empiris terkendali antara implementasi FSA berbasis State Pattern dan logika kontrol konvensional pada arsitektur game Three.js. Berdasarkan empat tujuan pengujian yang dirumuskan, temuan utama penelitian ini adalah: Pertama, pada kinerja komputasi runtime, FSA menanggung penalti biaya pembaruan kontroler yang konsisten 16–21 persen lebih tinggi akibat overhead pemanggilan metode terdelegasi dan evaluasi guard terpusat. Kedua, pada kualitas struktur kode, FSA mempertahankan isolasi kompleksitas modul secara stabil dan mematuhi Open-Closed Principle dengan meniadakan penulisan ulang modul eksisting saat penambahan perilaku baru, berbanding kontroler konvensional yang mengharuskan penggantian seluruh berkas dasar dan meningkatkan kompleksitas fungsi terburuk. Ketiga, pada skalabilitas populasi, kedua paradigma menunjukkan pola pertumbuhan linear $O(N)$ tanpa bukti degradasi superlinier pada rentang beban hingga 500 agen. Keempat, pada determinisme perilaku, hook siklus hidup State Pattern berhasil mengeliminasi divergensi status-animasi yang kerap muncul pada kontroler imperatif.

Temuan ini berlaku spesifik pada konfigurasi eksperimen yang diuji dalam lingkungan peramban web dan tidak membuktikan keunggulan universal FSA atas seluruh domain rekayasa *game*. Pemilihan di antara kedua pendekatan merupakan pertukaran rekayasa yang seimbang: kontroler konvensional direkomendasikan untuk purwarupa cepat atau game dengan perilaku sederhana demi efisiensi CPU mentah, sedangkan FSA State Pattern direkomendasikan untuk sistem game skala menengah-panjang yang menuntut integritas animasi dan skalabilitas fitur yang terprediksi. Penelitian lanjutan disarankan untuk: (1) menguji kinerja pada lingkungan peramban multi-platform (Chrome, Firefox, Safari) dengan akselerasi perangkat keras GPU terdedikasi, (2) mengevaluasi skalabilitas populasi masif (N > 1.000 agen) dengan paralelisasi Web Workers atau WebAssembly (Wasm), serta (3) membandingkan efisiensi runtime dan modularitas FSA terhadap arsitektur *Behavior Tree* (BT) dan *Hierarchical* FSM (HFSM) untuk perilaku kombinatorial kompleks.

REFERENSI

[1] WHATWG, “HTML Standard - Event loops.”

[2] Three.js, “Three.js - JavaScript 3D library, rel. r163.”

[3] Khronos Group, “WebGL 2.0 specification.”

[4] S. Sintaro *et al*., “Implementation and comparison in using state pattern on main character movement (case study: Pocong Jump video game version 1.0),” *BAREKENG: Jurnal Ilmu Matematika dan Terapan*, vol. 17, no. 2, pp. 955–968, 2023, doi: 10.30598/barekengvol17iss2pp0955-0968.

[5] R. Nystrom, *Game programming patterns*. Genever Benning, 2014.

[6] S. Rabin, Ed., *Game AI pro 3: collected wisdom of game AI professionals*. Boca Raton, FL, USA: CRC Press, 2017.

[7] Google, “TurboFan - optimizing compiler.”

[8] T. J. McCabe, “A complexity measure,” *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976, doi: 10.1109/TSE.1976.233837.

[9] M. H. Halstead, *Elements of software science*. New York, NY, USA: Elsevier, 1977.

[10] L. K. Alqutbhi, A. P. Sasmito, and D. Rudhistiar, “Implementasi finite state machine (FSM) dan line of sight (LoS) dalam perancangan game 2D petualangan Raden Panji Inu Kertapati,” *JATI (Jurnal Mahasiswa Teknik Informatika)*, vol. 9, no. 2, pp. 3542–3550, 2025, doi: 10.36040/jati.v9i2.13336.

[11] M. B. K. Bayu, U. M. H. Tamyiz, and L. S. Andar, “Penerapan finite state machine pada squid game berbasis desktop menggunakan Unity 3d,” *Jurnal Aplikasi dan Teori Ilmu Komputer*, vol. 7, no. 1, pp. 51–57, Mar. 2024, doi: 10.17509/jatikom.v7i1.60718.

[12] A. E. Adeniyi *et al.*, “Development of two dimension (2D) game engine with finite state machine (FSM) based artificial

intelligence (AI) subsystem," *Procedia Computer Science*, vol. 235, pp. 2996–3006, 2024, doi: 10.1016/j.procs.2024.04.283.

[13] M. Ç. Uluda\ugl\i and K. O\uguz, "Non-player character decision-making in computer games," *Artificial Intelligence Review*, vol. 56, no. 12, pp. 14159–14191, 2023, doi: 10.1007/s10462-023-10491-7.

[14] M. Kopel and T. Hajas, "Implementing AI for non-player characters in 3D video games," in *Lecture Notes in Computer Science*, 2018, pp. 610–619. doi: 10.1007/978-3-319-75417-8_57.

[15] H. Armanto, H. A. Rosyid, Muladi, and Gunawan, "Improved non-player character (NPC) behavior using evolutionary algorithm - a systematic review," *Entertainment Computing*, vol. 52, p. 100875, 2025, doi: 10.1016/j.entcom.2024.100875.

[16] F. F. Duarte, N. Lau, A. Pereira, and L. P. Reis, "A survey of planning and learning in games," *Applied Sciences*, vol. 10, no. 13, p. 4529, 2020, doi: 10.3390/app10134529.

[17] N. A. Mas'udi, E. M. A. Jonemaro, M. A. Akbar, and T. Afirianto, "Development of non-player character for 3D kart racing game using decision tree," *Fountain of Informatics Journal*, vol. 6, no. 2, p. 51, 2021, doi: 10.21111/fij.v6i2.4678.

[18] M. Iovino, J. Förster, P. Falco, J. J. Chung, R. Siegwart, and C. Smith, "Comparison between behavior trees and finite state machines," *IEEE Transactions on Automation Science and Engineering*, vol. 22, pp. 21098–21117, 2025, doi: 10.1109/TASE.2025.3610090.

[19] R. Sun, B. Chen, N. Xiang, and Z. Xu, "FSM vs. DRL: A comparative study of enemy behavior in fighting games," *IFAC-PapersOnLine*, vol. 59, no. 35, pp. 631–636, 2025, doi: 10.1016/j.ifacol.2025.12.549.

[20] J. Dirksen, *Learn Three.js*, 4th ed. Birmingham, U.K.: Packt Publishing Limited, 2023.

[21] G. Koulaxidis and S. Xinogalos, "Improving mobile game performance with basic optimization techniques in Unity," *Modelling*, vol. 3, no. 2, pp. 201–223, Mar. 2022, doi: 10.3390/modelling3020014.

[22] L. Lavazza, A. Z. Abualkishik, G. Liu, and S. Morasca, "An empirical evaluation of the 'Cognitive Complexity' measure as a predictor of code understandability," *Journal of Systems and Software*, vol. 197, p. 111561, Mar. 2023, doi: 10.1016/j.jss.2022.111561.

[23] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," *Computer*, vol. 27, no. 8, pp. 44–49, Aug. 1994, doi: 10.1109/2.303623.

[24] R. Mohamed, W. Al Adrousy, and S. Elmougy, "Toward the believability of non-player characters (NPC) movement in video games," *Fusion: Practice and Applications*, vol. 14, no. 1, pp. 66–80, 2024, doi: 10.54216/FPA.140106.

[25] G. N. Yannakakis, "Game AI revisited," in *Proceedings of the 9th ACM International Conference on Computing Frontiers*, 2012, pp. 285–292. doi: 10.1145/2212908.2212954.

[26] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and AI," *Robotics and Autonomous Systems*, vol. 154, p. 104096, Aug. 2022, doi: 10.1016/j.robot.2022.104096.